Fornite

+ field: type

+ getMyQueue():IQueue<Player>
+ setMyQueue(IQueue<Player>): void
+getConsoleTable():IHashTable<Integer,Player>
+setConsoleTable(IHashTable<Integer,Player>):void
+getPhoneTable():IHashTable<Integer,Player>
+setPhoneTable((IHashTable<Integer,Player>):void
+getPcTable():(HashTable<Integer,Player>
+setPcTable(IHashTable<Integer,Player>):void
+dequeue():void
+IsQueueEmpty():boolean
+enqueue(Player):void
+sizeQueue():int
+ addTable(Player, Integer): void
+searchTable(IHashTable<Integer,Player>, Integer):Player
+ addFirstPlayer(IHashTable<Integer, Player>):void
+addLowerLevel(Player, IHashTable<Integer,Player>):void
+addHighLevel(Player, IHashTable<Integer,Player>):void
+newSkillHigh(int):int
+newPlayerHash(int,int,int,IHashTable<Integer, Player>):Player
+newSkillLow(int):int
+generateRandom(int,int):int
+ readPlayers():void

1

-consoleTable   phoneTable   pcTable

1
-m

<<blind>>
K --> id
V ---> player

IHashTable <K,V>
<<interface>>

+ add(V, K): void

Player

+static final String PC="PC"
+static final String CONSOLE="Console"
+static final String PHONE="Phone"
- name: String
- ping: int
- geo: String
- skill: int
-int id
-plataform:String

+Player(String, int, String, int,int,String,Stack)
+getName():String
+setName(String):void
+getPing():int
+setPing(int):void
+getGeo():String
+setGeo(String):void
+getSkill():int
+setSkill(int):void
+getID():int
+setID(int):void
+getPlataform():String
+setPlataform(String):void
+getWeaponStack():weaponStack
+setWeaponStack(weaponStack):void

Weapon

+static final String RANGED="Ran
+static final String MELEE="Melee
-image: String
-name:String
-power:double
-type:String

+Weapon(String, String, double, S
+getImage():String
+setImage(String):void
+getName():String
+setName(String):void
+getPower():String
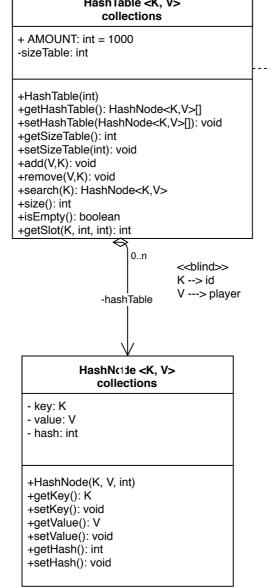+setPower(String):void
+getType():String
+setType(String):void

myQueue

<<blind>>
T --- > Player

nged"
e"

tring)

1

-weapons

<<blind>>
S ---> Weapon

HashTable <K, V>
collections

+ AMOUNT: int = 1000
-sizeTable: int

+HashTable(int)
+getHashTable(): HashNode<K,V>[]
+setHashTable(HashNode<K,V>[]): void
+getSizeTable(): int
+setSizeTable(int): void
+add(V,K): void
+remove(V,K): void
+search(K): HashNode<K,V>
+size(): int
+isEmpty(): boolean
+getSlot(K, int, int): int

<<blind>>
K --> id
V ---> player

+ add(V, K): void
+ remove(V, K): void
+ search(K):HashNode<K,V>
+ size(): int
+ isEmpty(): boolean
+ getSlot(K, int, int): int
+getHashTable(): HashNode<K,V>[]
+setHashTable(HashNode<K,V>[]): void
+getSizeTable(): int
+setSizeTable()int: void

0..n

<<blind>>
K --> id
V ---> player

-hashTable

HashNode <K, V>
collections

- key: K
- value: V
- hash: int

+HashNode(K, V, int)
+getKey(): K
+setKey(): void
+getValue(): V
+setValue(): void
+getHash(): int
+setHash(): void

IQueue \<T\>
\<\<interface\>\>

+ enqueue(T): void
+ dequeue(): void
+ search(T):void
+ size(): int
+ isEmpty(): boolean
+getFront(): Node\<T\>
+getBack(): Node\<T\>
+setFront(Node\<T\>): void
+setBack(Node\<T\>): void

\<\<blind\>\>
T --> Player

Queue \<T\>
collections

+ SIZE_ASS :int = 100
- front: Node\<T\>
-back: Node\<T\>

+ enqueue(T): void
+ dequeue(): void
+ search(T):void
+ size(): int
+ isEmpty(): boolean
+getFront(): Node\<T\>
+getBack(): Node\<T\>
+setFront(Node\<T\>): void
+setBack(Node\<T\>): void

1 — 1
- front   -back

\<\<blind\>\>
T --> Player

Node\<T\>
collections

- value: T

-previous

+ Node (T)
+getValue(): T
+setValue(T): void
+ getNext(): T
+ getPrevious(): Node\<T\>
+ setNext(Node\<T\>): void
+ setPrevious(Node\<T\>): void

1

1

1

Stack<S>
collections

+push(S): void
+pop(): void
+top(): S
+isEmpty(): boolean
+search(S): boolean
+size(): int
+getLast(): NodeStack<S>
+setLast(NodeStack<S>): void

IStack <S>
<<interface>>

+push(S): void
+pop(): void
+top(): S
+isEmpty(): boolean
+search(S): boolean
+size(): int

<<blind>>
S ---> Weapon

1

- last

<<blind>>
S ---> Weapon

NodeStack<S>
collections

- value: S

+ NodeStack (S)
+getNext(): NodeStack<S>
+setNext(NodeStack<S>): void
+getValue(): S
+setValue(S): void

1

1

-next

-next